

EffectiveMySQL.com

Its all about Performance and Scalability



Improving Performance with Better Indexes

Ronald Bradford
<http://ronaldbradford.com>

Oracle Open World
October 2011



EffectiveMySQL.com - Its all about **Performance** and **Scalability**

OBJECTIVE

Improve query performance,
therefore

Increase server throughput,
therefore

Reduce H/W to run your
application

ABOUT AUTHOR

RONALD BRADFORD

- 12 years with MySQL / 22 years with RDBMS
 - Senior Consultant at MySQL Inc (06-08)
 - Consultant for Oracle Corporation (96-99)
- 7 years presenting MySQL content
- All time top MySQL blogger
- Published author
- Oracle ACE Director

Available NOW
for consulting

<http://NYMySQLExpert.com>

<http://RonaldBradford.com>

EffectiveMySQL.com - Its all about **Performance** and **Scalability**



EffectiveMySQL.com - Its all about **Performance** and **Scalability**



EffectiveMySQL.com - Its all about **Performance** and **Scalability**

SOLUTION

Extra: Using Index

SQL REVIEW

Six step process

1. Capture
2. Identify
3. Confirm
4. Analyze
5. Optimize
6. Verify

CAPTURE

Not discussed
in detail this presentation

- General Query Log
- TCP/IP
- Connectors
- Application
- Proxy
- Plugin
- ...

1. Capture
2. Identify
3. Confirm
4. Analyze
5. Optimize
6. Verify

EffectiveMySQL.com - Its all about **Performance** and **Scalability**

CAPTURE

Not discussed
in detail this presentation

- General Query Log
- TCP/IP
- Connectors
- Application
- Proxy
- Plugin
- ...

All Queries?
or
Sample Queries?

1. Capture
2. Identify
3. Confirm
4. Analyze
5. Optimize
6. Verify

EffectiveMySQL.com - Its all about **Performance** and **Scalability**

CAPTURE EXAMPLE

Customer example

- Application Logging
 - All queries for a given function
- TCP/IP
 - Sample queries for a period of time

APPLICATION LOGGING EXAMPLE

PHP example

```
<?php
mysql_query('SELECT col1, col2 FROM table1
...
...
mysql_query('SELECT a,b,c FROM x INNER JOIN y ...
...
mysql_query('UPDATE table SET ...
?>
```

```
// Create wrapper function
function db_query($sql) {
    $rs = mysql_query($sql);
    return $rs;
}

// Global replace mysql_query with db_query
```

APPLICATION LOGGING EXAMPLE

```
function db_query($sql) {  
    $start_time = microtime(true);  
    $rs = mysql_query($sql);  
    $exec_time = microtime(true) - $start_time;  
    debug(format_time($exec_time) . ' ' . $sql . "\n");  
    return $rs;  
}  
  
function debug($str) {  
    if (DEBUG) echo $str;  
    return 0;  
}  
  
function format_time($time) {  
    return number_format($time, 8, '.', '');  
}
```

APPLICATION LOGGING EXAMPLE

```
0.00010109 SELECT ip FROM ...
0.00005198 SELECT name, value FROM ...
0.00005984 SELECT id, status, ...
0.17592907 SELECT g.id, c.id, ...
0.00047803 SELECT DISTINCT id_c FROM camp...
0.00741315 SELECT DISTINCT id_c FROM camp..
0.00058198 SELECT id_c FROM cr ...
0.00161815 SELECT id_c FROM cr ...
0.00032806 SELECT id_c FROM cr ...
0.00007200 SELECT DISTINCT id_a FROM arb ...
0.00005412 SELECT DISTINCT id_a FROM asw ...
0.00004697 SELECT id_adv FROM arw
0.00004601 SELECT id_adv FROM arw
0.00009012 SELECT gk.id, k.value ...
0.00009084 SELECT gk.id, k.value ...
0.00006318 SELECT gk.id, k.value ...
0.00005794 SELECT gk.id, k.value ...
0.00005603 SELECT gk.id, k.value ...
```

24 statements on page
Easy to spot worst query

APPLICATION LOGGING EXAMPLE

```
0.00010109 SELECT ip FROM ...
0.00005198 SELECT name, value FROM ...
0.00005984 SELECT id, status, ...
0.17592907 SELECT g.id, c.id, ...
0.00047803 SELECT DISTINCT id_c FROM camp...
0.00741315 SELECT DISTINCT id_c FROM camp..
0.00058198 SELECT id_c FROM cr ...
0.00161815 SELECT id_c FROM cr ...
0.00032806 SELECT id_c FROM cr ...
0.00007200 SELECT DISTINCT id_a FROM arb ...
0.00005412 SELECT DISTINCT id_a FROM asw ...
0.00004697 SELECT id_adv FROM arw
0.00004601 SELECT id_adv FROM arw
0.00009012 SELECT gk.id, k.value ...
0.00009084 SELECT gk.id, k.value ...
0.00006318 SELECT gk.id, k.value ...
0.00005794 SELECT gk.id, k.value ...
0.00005603 SELECT gk.id, k.value ...
```

24 statements on page
Easy to spot worst query

175 ms

7 ms

TCP/IP EXAMPLE

```
$ sudo tcpdump -i any port 3306 -s 65535 -x -nn -q -tttt  
-c 10000 | mk-query-digest --type tcpdump
```

```
# Profile  
# Rank Query ID           Response time Calls R/Call  Apdx  V/M  
# =====  
#      1 0xE5C8D4B9F7BDCCAF  0.5044 18.7%      1 0.5044 1.00  0.00 SELECT ..  
#      2 0x813031B8BBC3B329  0.4893 18.2%     23 0.0213 1.00  0.01 COMMIT  
#      3 0x04AB3BC1A33D3B48  0.4107 15.2%      1 0.4107 1.00  0.00 SELECT ..  
#      4 0xD15CA257BAF77DAF  0.3356 12.5%    321 0.0010 1.00  0.00 SELECT ..  
#      5 0x228B1F36C5EBCF1F  0.2177  8.1%       2 0.1089 1.00  0.22 SELECT ..
```

Capture SQL queries
executed for 1 second

<http://effectiveMySQL.com/article/mk-query-digest>

EffectiveMySQL.com - Its all about **Performance** and **Scalability**

IDENTIFY

Not discussed
in detail this presentation

- By Duration of time
- By Frequency of execution

1. Capture
2. Identify
3. Confirm
4. Analyze
5. Optimize
6. Verify

TCP/IP EXAMPLE

Frequency

```
//TODO
# Profile
# Rank Query ID           Response time Calls R/Call Apdx V/M  Item
# =====
#      1 0xE5C8D4B9F7BDCCAF 0.5044 18.7%      1 0.5044 1.00 0.00 SELECT ..
#      2 0x813031B8BBC3B329 0.4893 18.2%     23 0.0213 1.00 0.01 COMMIT
#      3 0x04AB3BC1A33D3B48 0.4107 15.2%      1 0.4107 1.00 0.00 SELECT ..
#      4 0xD15CA257BAF77DAF 0.3356 12.5%    321 0.0010 1.00 0.00 SELECT ..
#      5 0x228B1F36C5EBCF1F 0.2177  8.1%      2 0.1089 1.00 0.22 SELECT ..
```

Duration

CONFIRM

Not discussed
in detail this presentation

- Via
 - Application Logging
 - mysql client
 - SHOW PROFILES command

1. Capture
2. Identify
3. Confirm
4. Analyze
5. Optimize
6. Verify

CONFIRM

Not discussed
in detail this presentation

- Why confirm?
 - Watch & Isolate other factors
 - e.g.
 - Locking
 - Load
 - Caching

MYSQL CLIENT EXAMPLE

```
mysql> SELECT ...  
1 row in set (0.00 sec)
```

mysql client
10 millisecond precision



MYSQL CLIENT EXAMPLE

```
mysql> SELECT ...  
1 row in set (0.00 sec)
```

What you really want is?

```
mysql> SELECT ...  
1 row in set (0.008110 sec)
```



mysql client
microsecond precision

<http://j.mp/gM3Hb3>

PROFILING EXAMPLE

```
mysql> SET PROFILING=1;
```

```
mysql> SELECT SLEEP(1.0);
```

```
mysql> SELECT a,b,c FROM table LIMIT 1;
```

```
mysql> SELECT SLEEP(0.2);
```

```
mysql> SHOW PROFILES;
```

Query_ID	Duration	Query
1	1.00135300	SELECT SLEEP(1)
2	0.00026700	SELECT a,b,c FROM table ...
3	0.20215600	SELECT SLEEP(2)

Quick verification
microsecond precision

ANALYZE

- EXPLAIN
- SHOW CREATE TABLE
- SHOW INDEXES FROM
- INFORMATION_SCHEMA.TABLES
- SHOW TABLE STATUS LIKE
- EXPLAIN EXTENDED

1. Capture
2. Identify
3. Confirm
4. Analyze
5. Optimize
6. Verify

EffectiveMySQL.com - Its all about **Performance** and **Scalability**

EXAMPLE

```
mysql> EXPLAIN SELECT p.amt, p.reference
```

```
FROM      payment p
JOIN      invoice i ON i.inv_id = p.inv_id
WHERE     i.due_date = '2009-10-14'
AND       i.user_id = 1
AND       i.account_id = 10
AND       p.amt > 0
```

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	PRIMARY,u,a,d	u	4	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where

Format modified for
display purposes

EXAMPLE

```
mysql> SHOW CREATE TABLE wp_users \G
```

```
CREATE TABLE `wp_users` (  
  `ID` bigint(20) unsigned NOT NULL AUTO_INCREMENT,  
  `user_login` varchar(60) NOT NULL DEFAULT '',  
  `user_pass` varchar(64) NOT NULL DEFAULT '',  
  `user_nicename` varchar(50) NOT NULL DEFAULT '',  
  `user_email` varchar(100) NOT NULL DEFAULT '',  
  `user_url` varchar(100) NOT NULL DEFAULT '',  
  `user_registered` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',  
  `user_activation_key` varchar(60) NOT NULL DEFAULT '',  
  `user_status` int(11) NOT NULL DEFAULT '0',  
  `display_name` varchar(250) NOT NULL DEFAULT '',  
  PRIMARY KEY (`ID`),  
  KEY `user_login_key` (`user_login`),  
  KEY `user_nicename` (`user_nicename`)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8
```

Shows column
definitions and index
definitions

EXAMPLE

```
mysql> SHOW INDEXES FROM wp_posts;
```

...	Non_unique	Key_name	Seq_in_index	Cardinality	...
...	0	PRIMARY	1	<u>2606</u>	...
...	1	post_name	1	<u>2606</u>	...
...	1	type_status_date	1	<u>4</u>	...
...	1	type_status_date	2	6	...
...	1	type_status_date	3	2606	...
...	1	type_status_date	4	2606	...
...	1	post_parent	1	217	...
...	1	post_author	1	1	...
...					...

EXAMPLE

```
mysql> SHOW INDEXES FROM ...;
```

Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality
0	PRIMARY	1	id	A	100085
1	is_available	1	is_available	A	<u>4</u>
1	is_active	1	is_active	A	4
1	market	1	is_active	A	36
1	market	2	is_available	A	36
1	market	3	is_proposed	A	238

Shows uniqueness of
values in index

EXAMPLE

```
mysql> SELECT  table_schema, table_name, engine,  
               row_format, table_rows, avg_row_length,  
               (data_length+index_length)/1024/1024 as total_mb,  
               (data_length)/1024/1024 as data_mb,  
               (index_length)/1024/1024 as index_mb
```

```
FROM    INFORMATION_SCHEMA.TABLES
```

```
WHERE    table_schema=DATABASE()
```

```
AND      table_name = 'example'\G
```

```
table_schema: example  
table_name: example  
engine: MyISAM  
row_format: Dynamic  
table_rows: 260006  
avg_row_length: 2798  
total_mb: 700.20174026  
data_mb: 600.95564651  
index_mb: 99.24609375
```



Size of data on disk

EXAMPLE

```
mysql> SHOW TABLE STATUS LIKE 'wp_posts'\G
```

```
      Name: wp_posts
      Engine: MyISAM
      Version: 10
      Row_format: Dynamic
      Rows: 2649
      Avg_row_length: 2850
      Data_length: 7550800
      Max_data_length: 281474976710655
      Index_length: 259072
      Data_free: 0
      Auto_increment: 3586
      Create_time: 2011-02-02 12:04:21
      Update_time: 2011-03-19 11:42:12
      Check_time: 2011-02-02 12:04:21
      Collation: utf8_general_ci
```


EXAMPLE

```
mysql> EXPLAIN SELECT x.uuid, ..., y.uniqueid
        FROM table1 x, table2 y
        WHERE x.uuid=y.uniqueid;
```

id	select_	table	type	possible_	key	key_len	ref	rows	Extra
1	SIMPLE	x	ALL	NULL	NULL	NULL	NULL	29960165	
1	SIMPLE	y	eq_ref	PRIMARY	PRIMARY	194	func	1	Using where; Using index

EXAMPLE

```
mysql> EXPLAIN EXTENDED SELECT x.uuid, ..., y.uniqueid  
      FROM table1 x, table2 y  
      WHERE x.uuid = y.uniqueid;
```

```
mysql> SHOW WARNINGS;
```

Level: Note

Code: 10031

Message: select `schema`.`x`.`uuid` AS `uuid`, ...
`schema`.`y`.`uniqueid` AS `uniqueid` from
`schema`.`table1` `x` join `schema`.`table2` `y`
where (convert(`schema`.`x`.`uuid` using utf8) =
`test`.`y`.`uniqueid`

OPTIMIZE

Not discussed
in this presentation

- Adding Indexes
- Query simplification
- Eliminate queries
- MySQL configuration
- Schema design
- ...

1. Capture
2. Identify
3. Confirm
4. Analyze
5. Optimize
6. Verify

EffectiveMySQL.com - Its all about **Performance** and **Scalability**

THEORY

EXAMPLE

```
CREATE TABLE posts (  
  id          INT UNSIGNED NOT NULL AUTO_INCREMENT,  
  author      INT UNSIGNED NOT NULL,  
  status      VARCHAR(20) NOT NULL DEFAULT 'publish',  
  name        VARCHAR(100) NOT NULL,  
  content     TEXT NOT NULL,  
  comment_count INT UNSIGNED NOT NULL,  
  PRIMARY KEY (id),  
  KEY name (name)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

EXAMPLE

```
mysql> EXPLAIN
      SELECT COUNT(*) FROM posts
      WHERE author=2 AND status='draft' \G
```

```
      id: 1
select_type: SIMPLE
      table: posts
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 2649
      Extra: Using where
```

No Index used
i.e. Full Table Scan

EXAMPLE

```
mysql> ALTER TABLE posts ADD INDEX (author);  
mysql> EXPLAIN  
      SELECT COUNT(*) FROM posts  
      WHERE author=2 AND status='draft' \G
```

```
      id: 1  
  select_type: SIMPLE  
        table: posts  
        type: ref  
possible_keys: author  
          key: author  
    key_len: 4  
        ref: const  
       rows: 373  
   Extra: Using where
```

Adding an index

Less rows compared

EXAMPLE

```
mysql> ALTER TABLE posts ADD INDEX (author,status);  
mysql> EXPLAIN  
      SELECT COUNT(*) FROM posts  
      WHERE author=2 AND status='draft' \G
```

```
      id: 1  
  select_type: SIMPLE  
      table: posts  
      type: ref  
possible_keys: author  
      key: author  
  key_len: 66  
      ref: const,const  
      rows: 31  
  Extra: Using where; Using index
```

Adding a **better** index

Event less rows
compared

EXAMPLE

```
mysql> ALTER TABLE posts ADD INDEX (author)
mysql> EXPLAIN
SELECT COUNT(*) FROM posts
WHERE author=2 AND status='draft'
```

```
id: 1
select_type: SIMPLE
table: posts
type: ref
possible_keys: author
key: author
key_len: 4
ref: const
rows: 373
Extra: Using where
```

```
mysql> ALTER TABLE posts ADD INDEX (author,status)
mysql> EXPLAIN
SELECT COUNT(*) FROM posts
WHERE author=2 AND status='draft'
```

```
id: 1
select_type: SIMPLE
table: posts
type: ref
possible_keys: author
key: author
key_len: 66
ref: const,const
rows: 31
Extra: Using where; Using index
```

More columns

Less rows processed

COVERING INDEX

- Extra: Using Index
 - Does not mean using index
 - Means using **ONLY** the index
- All query columns are satisfied by index

COVERING INDEX

- Storage Engines matter
 - MyISAM Indexes
 - B-tree (PK & Secondary)
 - InnoDB Indexes
 - B+tree/Clustered (PK)
 - B-tree (secondary)

EXAMPLE

```
mysql> ALTER TABLE posts ENGINE=InnoDB;  
mysql> EXPLAIN  
      SELECT id FROM posts  
      WHERE author=2 AND status='draft' \G  
  
      id: 1  
select_type: SIMPLE  
      table: posts  
      type: ref  
possible_keys: author  
      key: author  
key_len: 66  
      ref: const,const  
      rows: 34  
Extra: Using where; Using index
```



Covering Index

EXAMPLE

```
mysql> ALTER TABLE posts ENGINE=MyISAM;
mysql> EXPLAIN
      SELECT id FROM posts
      WHERE author=2 AND status='draft' \G

      id: 1
select_type: SIMPLE
      table: posts
       type: ref
possible_keys: author
        key: author
    key_len: 66
       ref: const,const
       rows: 31
    Extra: Using where
```

Not a covering Index

PARENT/CHILD

EXAMPLE

```
CREATE TABLE invoice (  
  inv_id          INT UNSIGNED NOT NULL,  
  user_id         INT UNSIGNED NOT NULL,  
  account_id      INT UNSIGNED NOT NULL,  
  invoice_date    DATE NOT NULL,  
  due_date        DATE NOT NULL,  
  PRIMARY KEY pk (inv_id),  
  INDEX u (user_id),  
  INDEX a (account_id),  
  INDEX d (due_date)  
) ENGINE=InnoDB;  
  
CREATE TABLE payment (  
  pay_id          INT UNSIGNED NOT NULL AUTO_INCREMENT,  
  inv_id          INT UNSIGNED NOT NULL,  
  amt             DECIMAL (10,2) NOT NULL,  
  reference       VARCHAR(20) NOT NULL,  
  PRIMARY KEY pk (pay_id),  
  INDEX i (inv_id)  
) ENGINE=InnoDB;
```

Master/Child
Relationship

EXAMPLE

EXPLAIN

```
SELECT  p.amt, p.reference
FROM    payment p
JOIN    invoice i ON i.inv_id = p.inv_id
WHERE   i.due_date = '2009-10-14'
AND     i.user_id = 1
AND     i.account_id = 10
AND     p.amt > 0
```

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	PRIMARY,u,a,d	u	4	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where

Important Start

```
SHOW CREATE TABLE payment \G
```

```
...
PRIMARY KEY pk (pay_id),
INDEX i (inv_id)
) ENGINE=InnoDB;
```

EXAMPLE

```
SELECT  p.amt, p.reference
FROM    payment p
JOIN    invoice i ON i.inv_id = p.inv_id
WHERE   i.due_date = '2009-10-14'
AND     i.user_id = 1
AND     i.account_id = 10
AND     p.amt > 0
```

Identify additional
column

```
ALTER TABLE payment
DROP INDEX i, ADD INDEX i (inv_id, amt);
```

Add to index

EXAMPLE

Before

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	PRIMARY,u,a,d	u	4	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where

After

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	PRIMARY,u,a,d	u	4	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where

Zero Improvement

EXAMPLE

```
SELECT  p.amt, p.reference
FROM    payment p
JOIN    invoice i ON i.inv_id = p.inv_id
WHERE   i.due_date = '2009-10-14'
AND     i.user_id = 1
AND     i.account_id = 10
AND     p.amt > 0
```

Identify additional
column

```
ALTER TABLE payment
DROP INDEX i, ADD INDEX i (inv_id, amt, reference);
```

Add to index

EXAMPLE

Before

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	PRIMARY,u,a,d	u	4	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where

After

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	PRIMARY,u,a,d	u	4	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where; Using index

Covering Index



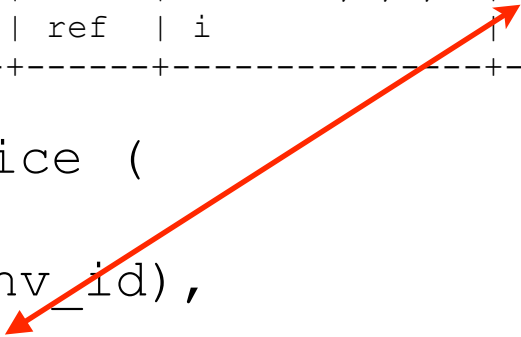
EXAMPLE

Second table in query

```
SELECT  p.amt, p.reference
FROM    payment p
JOIN    invoice i ON i.inv_id = p.inv_id
WHERE   i.due_date = '2009-10-14'
AND     i.user_id = 1
AND     i.account_id = 10
AND     p.amt > 0
```

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	PRIMARY,u,a,d	u	4	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where

```
CREATE TABLE invoice (
...
PRIMARY KEY pk (inv_id),
INDEX u (user_id),
INDEX a (account_id),
INDEX d (due_date)
) ENGINE=InnoDB;
```



EXAMPLE

```
SELECT  p.amt, p.reference
FROM    payment p
JOIN    invoice i ON i.inv_id = p.inv_id
WHERE   i.due_date = '2009-10-14'
AND     i.user_id = 1
AND     i.account_id = 10
AND     p.amt > 0
```

Identify additional
column



```
ALTER TABLE invoice
DROP INDEX u, ADD INDEX u (user_id, account_id);
```

Is account_id index
needed now?

EXAMPLE

Before

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	PRIMARY,u,a,d	u	4	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where; Using index

After

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	u	u	8	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where; Using index

Better Index Usage (4 + 4 bytes)

EXAMPLE

```
SELECT  p.amt, p.reference
FROM    payment p
JOIN    invoice i ON i.inv_id = p.inv_id
WHERE   i.due_date = '2009-10-14'
AND     i.user_id = 1
AND     i.account_id = 10
AND     p.amt > 0
```

Identify additional
column



```
ALTER TABLE invoice
DROP INDEX u, ADD INDEX u (user_id, account_id, due_date);
```

EXAMPLE

Before

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	u	u	8	1	Using where
1	SIMPLE	p	ref	i	i	4	6	Using where; Using index

After

id	select_type	table	type	possible_keys	key	key_len	rows	Extra
1	SIMPLE	i	ref	u	u	11	1	Using index
1	SIMPLE	p	ref	i	i	4	6	Using where; Using index

Covering Index
Better Index Usage (4 + 4 + 3 bytes)

VERIFY

- Quantifiable
- Reproducible

REAL WORLD

EXAMPLE

id	select_t	tab	key	key_	rows	Extra
1	PRIMARY	c	statu	1	74	Using where
1	PRIMARY	e	PRIMA	4	1	Using where
1	PRIMARY	g	campa	4	1	Using where
10	DEPENDEN	crb	id_ca	4	253	Using where
9	DEPENDEN	csb	pub_s	98	1	Using where
8	DEPENDEN	arb	id_ad	4	901	Using where
7	DEPENDEN	asb	pub_s	34	1	Using where
6	DEPENDEN	pm	id_adr	4	42	Using where
5	DEPENDEN	tgvl	searc	4	2	Using where
4	DEPENDEN	st	id_sc	4	7	Using where
4	DEPENDEN	t	PRIMA	4	1	Using where
3	DEPENDEN	k2	keywo	302	4	Using where
3	DEPENDEN	gk2	PRIMA	100	1	Using where
2	DEPENDEN	k1	keywo	302	3	Using where
2	DEPENDEN	gk1	PRIMA	100	1	Using where

BEFORE

175ms

REAL WORLD

EXAMPLE

id	select_t	tab	key	key_	rows	Extra
1	PRIMARY	e	statu	1	33	Using where; <u>Using index</u>
1	PRIMARY	c	adver	4	7	Using where
1	PRIMARY	g	campa	4	1	Using where
10	DEPENDEN	crb	id_ca	66	1	Using where
9	DEPENDEN	csb	pub_s	98	1	Using where
8	DEPENDEN	arb	id_ad	26	1	Using where
7	DEPENDEN	asb	id_ad	40	1	Using where; <u>Using index</u>
6	DEPENDEN	pm	id_adr	12	1	<u>Using index</u>
5	DEPENDEN	tgvl	searc	10	1	Using where; <u>Using index</u>
4	DEPENDEN	st	id_sc	4	7	Using where; <u>Using index</u>
4	DEPENDEN	t	PRIMA	4	1	Using where
3	DEPENDEN	k2	keywo	302	3	Using where; <u>Using index</u>
3	DEPENDEN	gk2	PRIMA	100	1	Using where
2	DEPENDEN	k1	keywo	302	2	Using where; <u>Using index</u>
2	DEPENDEN	gk1	PRIMA	100	1	Using where

AFTER

10ms

REAL WORLD

EXAMPLE

id	select_t	tab	key	key_	id	select_t	tab	key	key_	Extra
1	PRIMARY	c	statu	1	1	PRIMARY	c	adver	4	Using where
1	PRIMARY	e	PRIMA	4	1	PRIMARY	e	statu	1	Using where; Using in
1	PRIMARY	g	campa	4	1	PRIMARY	g	campa	4	Using where
10	DEPENDEN	crb	id_ca	4	10	DEPENDEN	crb	id_ca	66	Using where
9	DEPENDEN	csb	pub_s	98	9	DEPENDEN	csb	pub_s	98	Using where
8	DEPENDEN	arb	id_ad	4	8	DEPENDEN	arb	id_ad	26	Using where
7	DEPENDEN	asb	pub_s	34	7	DEPENDEN	asb	id_ad	40	Using where; Using in
6	DEPENDEN	pm	id_adr	4	6	DEPENDEN	pm	id_adr	12	Using index
5	DEPENDEN	tgv	searc	4	5	DEPENDEN	tgv	searc	10	Using where; Using in
4	DEPENDEN	st	id_sc	4	4	DEPENDEN	st	id_sc	4	Using where; Using in
4	DEPENDEN	t	PRIMA	4	4	DEPENDEN	t	PRIMA	4	Using where
3	DEPENDEN	k2	keywo	302	3	DEPENDEN	k2	keywo	302	Using where; Using in
3	DEPENDEN	gk2	PRIMA	100	3	DEPENDEN	gk2	PRIMA	100	Using where
2	DEPENDEN	k1	keywo	302	2	DEPENDEN	k1	keywo	302	Using where; Using in
2	DEPENDEN	gk1	PRIMA	100	2	DEPENDEN	gk1	PRIMA	100	Using where

Only added columns to existing indexes.
No Code Changes.
No Configuration Changes

Sometimes a Covering Index is ineffective

Partial Column Index

EffectiveMySQL.com - Its all about **Performance** and **Scalability**

EXAMPLE

```
+-----+-----+-----+-----+-----+-----+-----+
| id | select_t | tab | key      | key_ | rows | Extra      |
+-----+-----+-----+-----+-----+-----+-----+
...
| 10 | DEPENDEN | crb | id_ca    | 4     | 253 | Using where |
....
```

```
+-----+-----+-----+-----+-----+-----+-----+
| id | select_t | tab | key      | key_ | rows | Extra      |
+-----+-----+-----+-----+-----+-----+-----+
...
| 10 | DEPENDEN | crb | id_ca    | 66   | 1  | Using where |
...
```


EXAMPLE

```
...  
NOT EXISTS (  
    SELECT 1  
    FROM    crb  
    WHERE   crb.id = p.id AND crb.value = 'xyz')  
...
```

```
CREATE TABLE crb(  
    id          INT UNSIGNED NOT NULL,  
    value       VARCHAR(100) NOT NULL,  
    ....
```

```
    INDEX (id, value(20))  
) .. CHARSET=utf8;
```

66 bytes = 4 + (20*3) + 2

RE-OPTIMIZE

- Change data types
- Simplify query
- Removed normalization
- Other secrets ...

Optimizing SQL is an **iterative** process

Repeat as necessary

- 175ms to 10ms to 3ms
- Happy client is now more happy

VERY, VERY IMPORTANT

Cardinal Sin

- Indexes affect all queries on table

CONCLUSION

- 6 steps to successful SQL review
- Optimization is an **iterative** process
- Rows, storage engines, configuration & MySQL version **affect** results over time
- Indexes are not the **only** optimization
- Indexes are not the **best** optimization

EM=PS'n

Ronald Bradford

<http://effectiveMySQL.com>